AFRL-IF-RS-TR-2006-139
Final Technical Report
April 2006

LOW POWER COMPUTING IN DISTRIBUTED SYSTEMS

State University of New York at Binghamton

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** APRIL 2006	**3. REPORT TYPE AND DATES COVERED** Final Jun 2006 – Aug 2005

4. TITLE AND SUBTITLE
LOW POWER COMPUTING IN DISTRIBUTED SYSTEMS

5. FUNDING NUMBERS
C - FA8750-05-1-0027
PE - 62702F
PR - 4519
TA - QQ
WU - IU

6. AUTHOR(S)
Qinru Qiu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
State University of New York at Binghamton
P. O. Box 600
Binghamton New York 13902

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTC
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-139

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Daniel J. Burns/IFTC/(315) 330-2335 Daniel.Burns@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
Three research topics have been investigated: (1) Energy aware programming techniques for embedded systems; (2) Dynamic power management of a sensor node with periodic incoming tasks; and (3) Resource management of a sensor network using a distributed Genetic Algorithm. The research shows that (1) Certain simple program transformations can significantly improve the performance and energy dissipation of the embedded processor, which is the core part of a sensor node or a mobile computing device; (2) The traditional task scheduling algorithm does not work well with power management in a sensor node. Two heuristic algorithms are proposed which give lower power consumption than the traditional algorithms. (3) The power management problem in a distributed system may be formulated as an optimization problem and solved using a distributed GA. The performance and energy of a distributed GA is determined by its configuration parameters such as: the sub-population size, the number of processors, the length of epoch, and the number of migrating individuals. Their relations with the convergence speed and the energy dissipation is analyzed in this report. This effort leveraged a distributed (parallel) Island Model Genetic Algorithm code developed in IFTC that was adapted to the 3rd problem and run on a cluster computer in IFTC this summer. A follow-on grant will study a fuller version of the problems with more realistic scenarios.

14. SUBJECT TERMS
Power management policy, distributed systems, sensor networks, genetic algorithm

15. NUMBER OF PAGES
20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

Due to the fast development of information technology, the networked distributed system will gradually replace the conventional centralized system of a single processing element. It is a vision of the future that large numbers of low cost smart mobile devices will be integrated into the daily life of ordinary people. Accumulated, they provide the information processing capability that is equivalent to a high performance processing station. The emerging concept of Ambient Intelligence reflects such a vision [1]. The recent developments of sensor networks [2], wearable computing systems [3] and cognitive systems validated the feasibility of this vision. The common features of these systems can be summarized as the following:

1. The system consists of multiple heterogeneous networked processing nodes.
2. Each node is battery powered; therefore, it has limited energy resources. How to process more tasks before the depletion of the battery energy is a big challenge.
3. The node must be very low cost so that it can be deployed in large quantities. To reduce the cost, the node is built with limited memory spaces and low cost embedded processors. Examples of those embedded processors are Intel XScale processor, IBM PowerPC processor, etc. The special architecture of the node imposes new requirements to the software program that is running on it.
4. Large computing or sensing tasks are done by multiple nodes collaboratively. This introduces new requirements on how to reduce the communication overhead.

To achieve energy efficiency in a distributed system, low power techniques at both node level and system level are studied.

At node level, the target is to minimize node energy dissipation while providing the requested service. In general, a node provides three types of services: sensing, computing and communication. Much work has been done on low power computing and communication. There are low power techniques in almost all aspects of software and hardware design, from energy aware programming, algorithm transformation and hardware-software partitioning, to logic synthesis, transistor sizing and buffer insertion. References [4] and [5] give good review on the existing low power techniques in VLSI chip and system design. Similarly there are low power techniques in every communication layer, from single processing to routing algorithms [6]~[9]. Besides low power computing and low power communication, power management is another widely used system level low power technique that reduces the wasted energy by turning off or slowing down the idle or under utilized device [10]~[12].

At system level, in order to provide stable and reliable service, it is desirable all of the nodes are fully functional. Therefore the network lifetime is usually considered as the shortest lifetime of any of the nodes in the network. For most state-of-the-art mobile distributed systems, one of the major constraints on node lifetime is its battery capacity. If the initial battery capacities are the same for all of the nodes, the battery lifetime is inversely proportional to the battery discharging rate. Hence, it is important to distribute tasks among the nodes evenly so that each node discharges its battery at approximately the same rate and has approximately the same lifetime.

The theme of my work during this summer is low power computing in distributed systems. Three topics have been investigated.

(1) Energy aware programming techniques for embedded systems

(2) Dynamic power management of a sensor node with periodic incoming tasks

(3) Resource management of a sensor network using a distributed Genetic Algorithm

The selection of these topics is based on the above mentioned special requirements of a distributed system, especially a sensor network. At the node level, special architectures of an embedded system and its impact on the performance and power efficiency of a software program have been studied. Several power management policies for a typical sensor node with periodic tasks have been proposed. At the system level, the algorithm that distributes tasks among multiple nodes has been considered. This problem is formulated as a resource optimization problem and is solved using a distributed Genetic Algorithm (GA). The performance and energy efficient implementation of the GA have been investigated.

2 ENERGY AWARE PROGRAMMING FOR EMBEDDED SYSTEM

Given an embedded system and two software programs A and B, which program consumes more power? This is a typical concern in a system engineer's mind when designing a sensor node or a portable computing device. Similar questions are: Is software power estimation a necessary step in the design flow? Is there any trade-off between power and performance in software design? Can we use floating point data, or will it consume more power? Next we are going to give some answers to these questions.

In reference [13], Intel 486DX2 and Fujitsu SPARClite 934 processors are analyzed to determine instruction level power consumption. The energy of each instruction is modeled as the sum of a base energy cost and a small variance due to data variation and inter-instruction effects, which is usually less than 10% of the base cost. Figure 1 gives the base power consumption of some of the most commonly used instructions. It shows that instructions with different functionality and different addressing modes have very different power consumptions. For example, the move operation with indirect address mode has the highest power consumption, which is almost 2X that of the NOP operation. Figure 2 gives the number of clock cycles for each instruction. The base energy is the product of the current, the time, and the supply voltage level. The relation between the instruction energy and the instruction execution time is given in Figure 3. In general, the energy dissipation increases as the execution time increases.

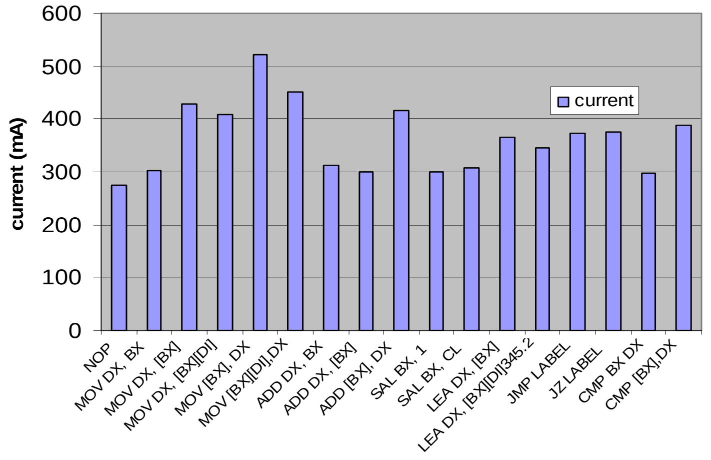


Figure 1 Instruction power consumption for the 486DX2

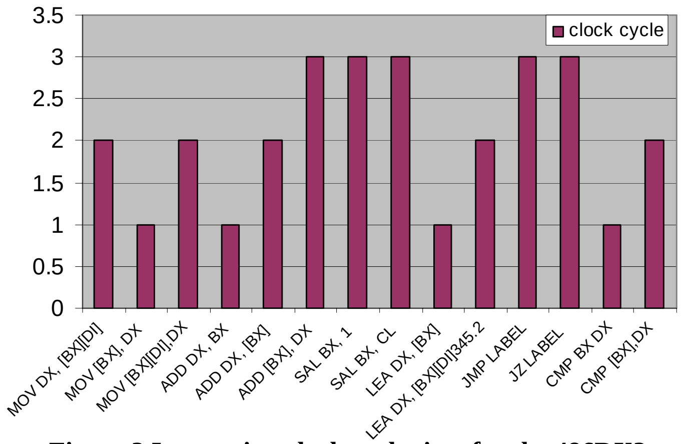


Figure 2 Instruction clock cycle time for the 486DX2

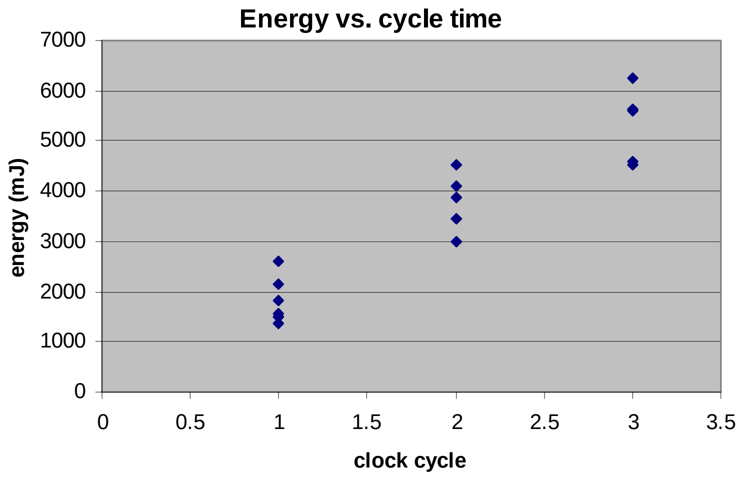


Figure 3 Energy vs. instruction cycle time

Reference [14] presents some results on instruction level energy profiling for the StrongARM SA1100 processor. It shows that the average current (power consumption) of different instructions has about a 38% variance. However, the current (power) consumption in program is much less. Figure 4 gives the current consumption of different programs running at different clock speeds. It shows that the power consumption of a processor is more determined by its clock frequency than the program that is running on it. Since different programs have almost the same power consumption, the difference of their energies is dominated by their runtimes.

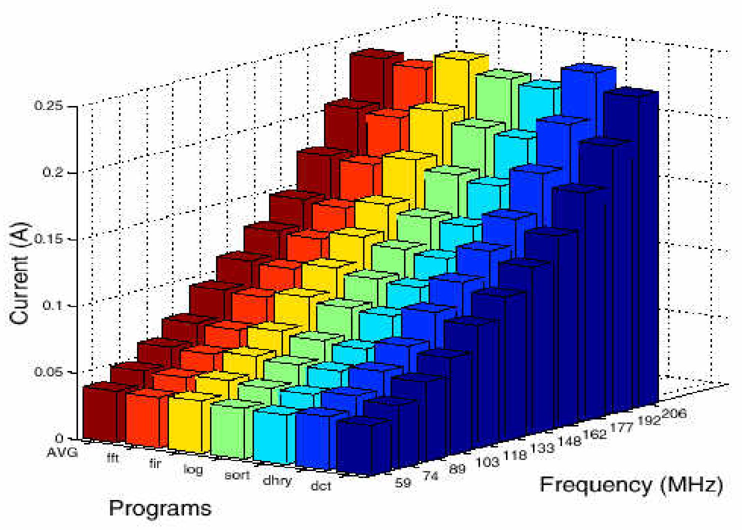


Figure 4 Program current consumption as a function of program and frequency

From reference [13] and [14] we know that the most efficient way to reduce the energy dissipation of a software program is to reduce its runtime. Reducing the power consumption by changing a program is more difficult and less effective. Therefore it should be considered as the second option.

Both of the works in [13] and [14] are based on microprocessors other than the Xscale processor. The Xscale processor [17] from Intel is one of the most popular processors designed for low power and high performance applications. It has been adopted in embedded systems such as the Stargate from Crossbow [15] and the PASTA system from USC ISI [16]. We are interested to find out the power and performance of software program running on the Xscale processor and find the appropriate energy aware programming style.

The information on program power consumption can be obtained either by software based power estimation or by hardware based current measurement. Some existing program power estimation tools include PowerAnalyzer [18], SimplePower [19], and JouleTrack [20]. However, they need detailed CPU architecture and design information for accurate estimation. This information is usually not available. Hardware based current measurement is used in our experiment. Figure 5 shows the experiment setup. The current consumption of the Stargate board is measured by an Agilent digital multimeter 34401A. The digital multimeter is connected with the PC for data recording and analysis. The sampling rate is 2.5kHz. Three experiments were performed to compare different programming styles. The results are presented below.

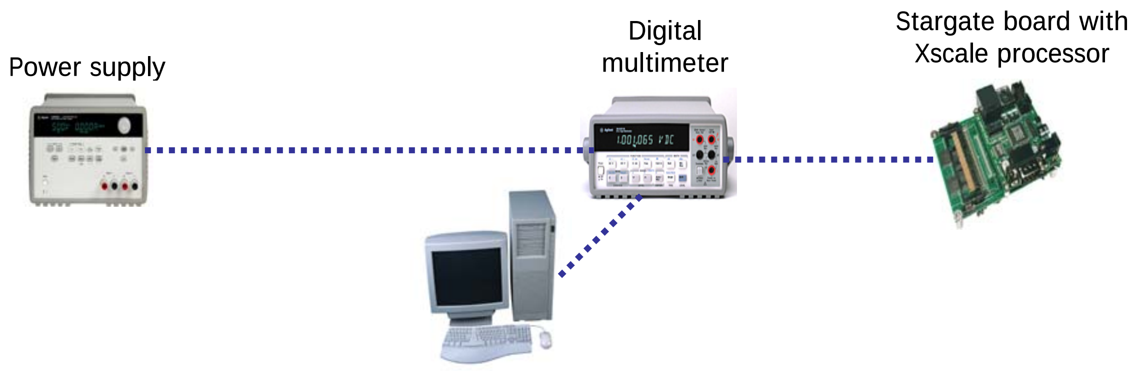


Figure 5 Experiment setup

2.1 Floating point operation vs. integer operation

In the first experiment, we compared the performance and power consumption of floating point operations to integer operations. Figure 6 gives the runtime and the average current consumption of 115 floating point and integer addition and multiplication operations. The average current consumption of the processor is about 0.14A when it is idle. The average runtime of a floating point operation is approximately 250 times slower than the corresponding integer operation. Further, the average current consumption of a floating point operation is approximately 20% higher than that of an integer operation. Overall, the energy consumption of a floating point operation is approximately 300 times higher than that of an integer operation. The reasons for this phenomenon are that the XScale processor does not have a hardware floating point unit, and that software based floating point calculation is time consuming.

The experimental results show that floating point operations should be avoided in the software design of an embedded system. If we can transform the floating point operation into integer operation by data scaling, we can reduce both the runtime as well as the energy dissipation of the program.

Operation	Run time (s)
115 floating point "+"	3.940
115 integer "+"	0.015
115 floating point "*"	4.365
115 integer "*"	0.020

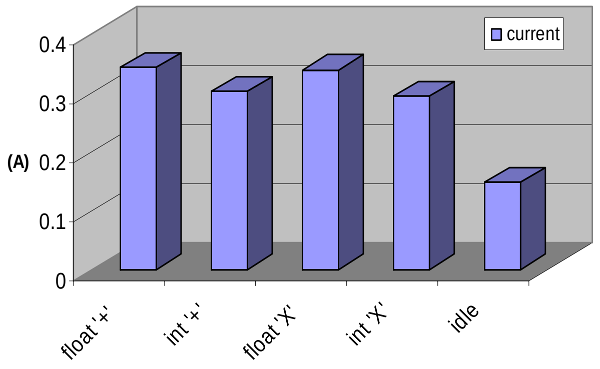


Figure 6 Performance and power consumption of floating point operation vs. integer operation

2.2 Library math function vs. user implemented math function

In the second experiment, we compared two implementations of the calculation of 3^a where a is an integer number. The first implementation uses the math library function *pow*(3, *a*) and the second implementation simply uses a loop of multiplications. Table 1 compares the two implementations in terms of runtime, current consumption, and energy dissipation, for values of a from 5 to 20. The results show that the loop based program consistently outperforms the library function based program. This is because the library function *pow*(*x*, *y*) contains some codes to handle the cases that the x or y or both are floating point values. Such processing is redundant if we knew that a is an integer number.

Table 1 Library function vs. user implementation of 3^a

	pow(3, *a*)			*o*=3;**for** *i*=1 to *a-1;* {*o*=*o**3;}		
	Current	Runtime	Energy	Current	Runtime	Energy
a=5	0.330	3.45	1.138	0.305	0.30	0.092
a=10	0.330	3.45	1.138	0.310	0.55	0.171
a=20	0.330	3.45	1.138	0.310	1.02	0.316

2.3 DNA code search program

In the last experiment, we modified the DNA code search program that was developed at AFRL/IFTC and port it to the Stargate system. Most of the variables in the DNA search program are integer variables but floating point operations are used in the process of generating random numbers. For example, in the original program, to generate a number x which is a random value chosen from 0 to *max*, the following calculation is used,

$x = (float)\ rand(\ ) / \mathrm{RAND\_MAX} * max$,

where RAND_MAX is equal to (2^{31} - 1) which is the largest integer number that can be represented by 32 bits. Floating point operation must be used here, otherwise the result will be 0, because "*rand*( )/RAND_MAX" is less than 1. However, by simple transformation, the calculation can be

$x = rand(\ ) / (\mathrm{RAND\_MAX} / max)$.

Since all the data in the above equation are integer variables, the divisions will be implemented as floored division in the calculation. The actual value of x can be written as:

$x = \lfloor rand(\ ) / \lfloor \mathrm{RAND\_MAX} / max \rfloor \rfloor$.

Because most of the times *max* is much less than RAND_MAX and x is much larger than 1, such approximation is acceptable.

Most of the floating point operations in the DNA code search program can be converted into integer operations by the above mentioned transformation. One exception is in the selection procedure used before mating. In this procedure, a floating point operation is used to calculate the probability of selection. In order to use integer operation, the value of the mating probability is scaled up 1000 times.

Our experimental results show that the runtime of the modified program is about 0.3% less than the original program. Both of the two programs have the current consumption 0.29A, which is about the same as the integer addition/multiplication operation reported in section 2.1. This indicates that the original program is in fact dominated by integer operations, hence its performance and energy has very little room for improvement by simply replacing the floating point operations with corresponding integer operations. The result also tells us that these types of program transformations will be more useful in DSP applications characterized by heavy usage of floating point operations, rather than in a distributed GA application that is dominated by integer operations.

3 POWER MANAGEMENT FOR SENSOR NODE WITH PERIODIC INCOMING TASKS

Dynamic power management – which refers to selective shut-off or slow-down of system components that are idle or underutilized – has proven to be a particularly effective technique. The problem of finding a power management scheme (or policy) that minimizes power dissipation under performance constraints is of great interest to system designers. A simple and well-known heuristic policy is the “time-out” policy, which is widely used in today’s portable computers. In the “time-out” policy, one component will be shut down after it has been idle for a certain amount of time. The predictive system shutdown approach in [21][22] tries to achieve better power-delay trade-off by predicting the “on” and “off” time of the component. This prediction approach uses a regression equation based on the component’s previous “on” and “off” time to estimate the next “turn-on” time, such that the component can be turned on immediately before the request comes. Therefore, the system performance can be improved. However, this method is only applicable to few cases in which the requests are highly correlated. A power management approach based on a Markov decision process has been proposed in [23]. The system is modeled as a discrete-time Markov decision process by combining the stochastic models of its components. Once the model and its parameters are determined, an optimal power management policy can be obtained to achieve the best power-delay trade-off for the system. The work of [24] overcomes the shortcomings of [23] by introducing a new system model (as well as component models) based on the continuous-time Markov decision process. In [24], a power-managed system is modeled in the continuous-time domain, which is closer to the situation encountered in practice; the component models are simpler and can accurately model many realistic applications.

Almost all of the previous power management techniques assume that tasks occur randomly, which is typical in many systems which have frequent user interactions. However, there are a large number of systems that work with little user interaction. The task incoming time for these systems may be very deterministic and periodic. For example, a sensor node is such a system. Most of the time, a sensor node senses, processes, communicates and records information periodically without user interaction. It is obvious that such system cannot be modeled as a Markov process, and stochastic power management cannot be applied. Furthermore, even though we know exactly when the next task will come in, predictive power management policy will have a poor performance if executions of the tasks are not scheduled carefully, as we will show in an example later. The key for effective power management in a system with periodic incoming tasks is to schedule the execution of the tasks to maximize the usage of low power modes.

A sensor node usually consists of one sensing and data acquisition module and several processing modules. The block diagram of a typical sensor node is shown in Figure 7 (b). The sensing and data acquisition module samples data periodically. Each data sample goes through several operations. An example of the data flow graph is given in Figure 7 (a). A vertex in this graph represents an operation. An edge from vertex v_1 to vertex v_2 indicates that the output data from v_1 is the input data to v_2. There is an input queue and an output queue associated with each operation. The input data of an operation may be processed immediately after it comes in or it may be buffered and processed in a burst mode. Given two vertices v_1 and v_2, assume that there is an edge from v_1 to v_2. Because the input queue of v_2 is also the output queue of v_1, only one queue need to be physically implemented. Here we assume that only the input queue will be implemented. Furthermore, if v_1 is executed by module m_1 and the v_2 is executed by module m_2, then the input queue of v_2 is implemented as a buffer of module m. However, it is called the input queue of module m_2. Note that one module may have one or multiple input queues depending on how many operations are executed by this module.

One of the special properties of the tasks in a sensor node is that the processing time is much shorter than the time between incoming tasks. Traditional scheduling algorithms such as ALAP, ASAP, and List algorithms [25] do not work in this situation when combined with power management. Based on our research, two heuristic algorithms are proposed, which give better results than those traditional scheduling algorithms. Before introducing the heuristic scheduling algorithms, some notations need to be defined first.

$q_{i,m}$: The ith input queue of module m.

tp_s: The processing time of task s.

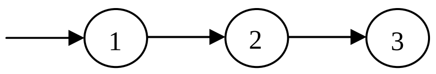
r_x: The sample time of data x

td_s: The maximum latency of processing up to step s.

$D_{x,s}$: The deadline for data x to finish step s processing. $D_{x,s} = r_x + td_s$

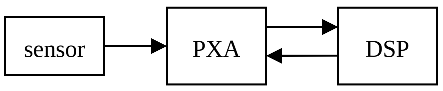
1. *ALAP-BURST* Given a module i. Assume that it has several input queue and q_i is associated with the ith step of processing. The first data x in q_i will be processed at time $r_{x,i} + td_i - tp_i$. After that, the rest of the data in q_i will be processed until the queue is empty.
2. *ALAP-DEPLETION* Given a module i with multiple input queues. The module processes data x, which is the first data in one of the input queues, at time t if $t = r_{x,i} + td_i - tp_i$. After that, the data in all of its input queues are processed until all of the queues are empty.

Next we are going to give an example that compares the two heuristics with ALAP scheduling. In all three cases, the predictive power management policy will be used.



$$\begin{tikzcd} {} \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \end{tikzcd}$$

(a) task flow graph



(b) system hardware configuration

Power/ energy	Active (w)	Idle (w)	Switching (J)
PXA	5.5	0	3
DSP	4	0	3

(c) power and switching energy

Task	Resource	*tp*	*td*
1	PXA	0.1	3
2	DSP	0.2	4
3	PXA	0.1	5

(d) task time information

Figure 7 Hardware and functionality of a sensor node

Figure 7 gives the hardware configuration and the task flow of a sensor node. The sensor consists of three modules. A sensing and data acquisition module, which is denoted as “sensor” in the system block diagram; a general purpose control and processor, which is implemented using an Intel PXA processor; a DSP processor, which is used for advanced data processing. Each data sampled by the sensing module goes through three operations sequentially, which are denoted as operation 1, 2, and 3. Operations 1 and 3 are executed by the PXA and operation 2 is executed by the DSP. Therefore, there is data flow from the sensing and data acquisition module to the PXA. There is also data flow from the PXA to the DSP and vise versa. The arrows in the block diagram represent the directions of data flow. The input queue of operation 1 is a buffer in the sensing module. The input queue of operation 2 is a buffer in the PXA and the input queue of operation 3 is a buffer in the DSP. Hence, the PXA has two input queues, the DSP has one input queue and the sensor has no input queue.

Timing information for these tasks is given in Figure 7 (d). The first column of the table gives the index of the task, the second column gives the name of module that is selected to run the task, the third column gives the processing time of the task (t_p) and the last column gives the maximum latency time of the task at each step (t_d). For example, if a data sample is taken at time t, then it must finish operation 1, 2 and 3 before t+3, t+4 and t+5, respectively. Assume that both the PXA and DSP can be put into sleep mode. Figure 7 (c) gives the power consumption of the PXA and DSP when they are active or sleeping as well as the energy cost for switching between these two states. Since most of sensing modules have very low power consumption, we will not consider power management of the sensing module.

Figure 8 gives the processing activity on the PXA and DSP using three heuristic scheduling algorithms. Using the traditional ASAP algorithm, the PXA and DSP will be turned on and off once every second. Hence, the average power consumption of the system will be 7.9W. Note that the PXA will not be turned off immediately after finishing operation 1 because the switching cost is higher than the cost of keeping the power on. Using ALAP-BURST, the PXA will be turned on and off twice every 4 seconds, while the DSP will be turned on and off once every 4 seconds. Therefore, the average power consumption of the system is 4.15W. Using ALAP-DEPLETION, both the PXA and DSP will be turned on and off twice every 6 seconds. Therefore, the average power consumption will be 3.9W. Note that the ALAP-DEPLETION performs better than ALAP-BURST because the DSP and the PXA has approximately the same switching energy. If the DSP has higher switching energy, then the ALAP-DEPLETION algorithm will give larger average power than the ALAP-BURST. This is the reason that we present both algorithms. The selection between these two should be made at the end of the design time when the hardware has been characterized.

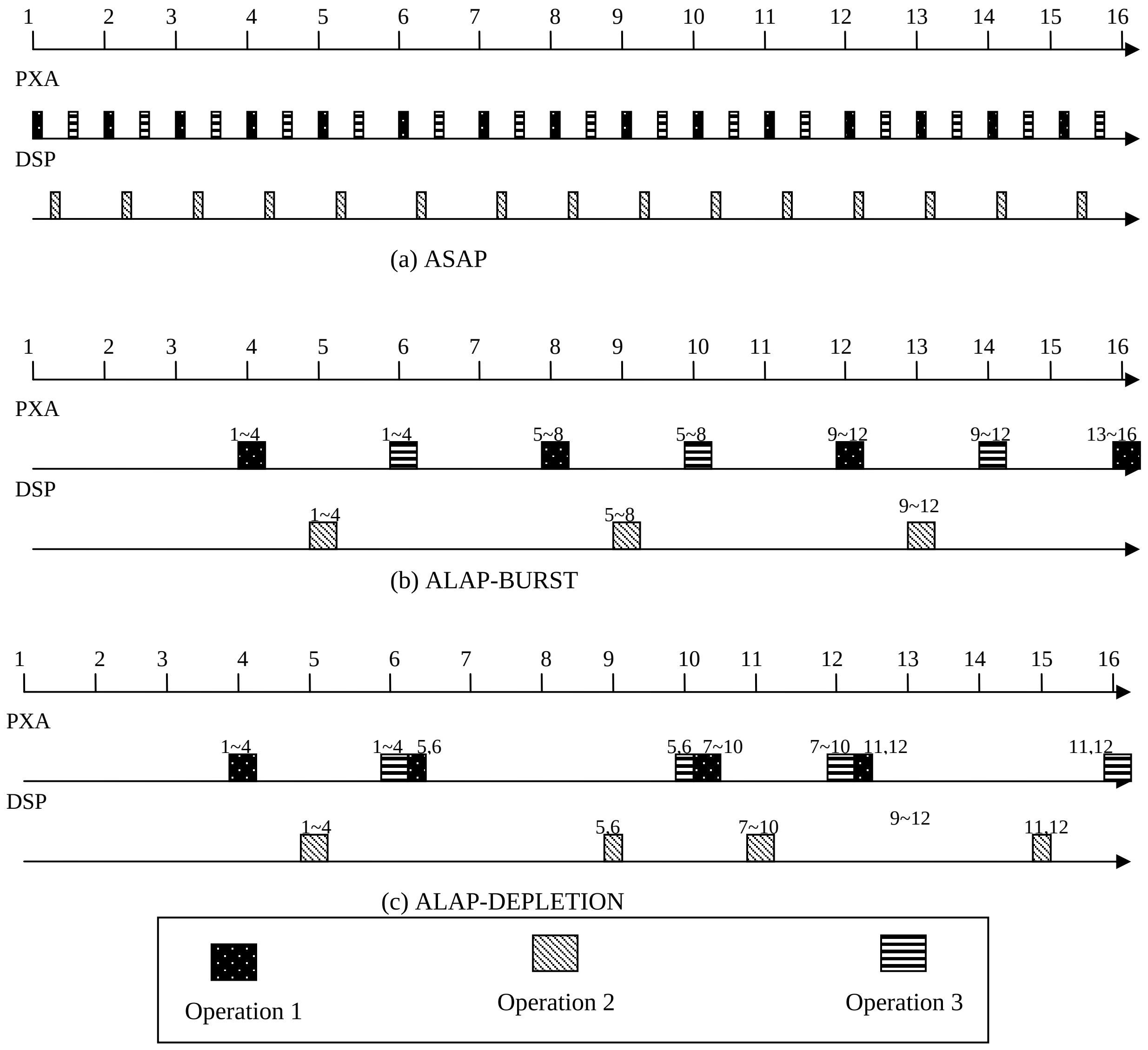
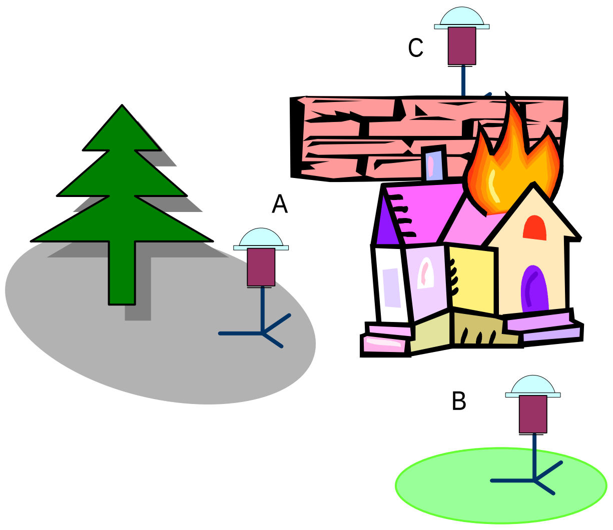


Figure 8 Comparison of heuristic scheduling algorithms

There are other scheduling algorithms that outperform these two algorithms. There are other constraints such as the buffer size constraint, the setup time constraint, etc. In our future work we intend to further improve the scheduling algorithm by considering sensor data sampling requirements for various situations.

4 RESOURCE MANAGEMENT OF A SENSOR NETWORK USING A DISTRIBUTED GENETIC ALGORITHM

Consider a sensor network with three sensor nodes which are deployed to detect fire in a building, as illustrated in Figure 9 (a). Node C is located behind a wall, node B is located relatively far from the building, node A is close to the building but it is covered by a tree. Each sensor has three detection and processing functions, smoke detection, temperature measurement, and light sensing. Each of these functions provides some information about whether the building is on fire or not. However, the probability of detection varies with node location. For example, node C is behind a wall, therefore its capability of detecting a fire using light sensing is weakened. We assume that different sensing functions have different energy dissipations in the different locations. Figure 9 (b) gives the detection probability and the energy dissipation for all of the three detection functions for the sensors in the three different locations. The user requirement is to detect the fire with >95% accuracy. This can be achieved by using light sensing on node A. However, the energy dissipation using this function is much higher than using the other functions. Another choice is to use temperature detection on node A and C and smoke detection on node B. Combined together, they provide the required detection probability of >95% with a total energy of 0.4J, which is much less than using light detection.



(a) sensor network

Task	Node	Detection Prob.	Energy (J)
Smoke	A	0.7	0.2
temperature √	A	0.6	0.1
Light	A	0.95	0.8
Smoke √	B	0.8	0.2
Temperature	B	0.6	0.2
Light	B	0.5	0.8
Smoke	C	0.7	0.2
Temperature √	C	0.6	0.1
Light	C	0.6	0.8

(b) node information

Figure 9 Motivational example

This example shows that in a collaborative distributed system, the tasks should be carefully selected and assigned to different nodes to minimize the total energy dissipation.

Resource management is defined as the process that assigns tasks to different nodes, schedules their start times, and decides the level of service quality. This determines the resource usage for running these tasks, e.g. the energy dissipation and required communication bandwidth. The execution of each task represents a positive gain when measuring or quantifying the performance of the system. It also associates a cost, which may be the energy dissipation, and/or the communication bandwidth, etc. The objective of resource management is to maximize the gain while minimizing the cost. To formulate the resource management problem into an optimization problem, we assume that the system and each of its nodes have certain levels of awareness of their environments, including the characteristics and requirements of the events that are happening, the capabilities of neighbors, node locations, performance specifications, constraints, and priorities, etc. to support decisions about how to carry out its operation in an optimal manner.

A Genetic Algorithm (GA) is a stochastic search technique based on the mechanism of natural selection, recombination, and mutation. It starts with an initial population of individuals, i.e. a set of randomly generated candidate solutions. The solutions are represented by *chromosomes*, which are collections of numbers or symbols that map onto parameters of the problem. Individuals are evolved from *generation* to generation, with *selection*, *crossover*, and *mutation* operators that provide an effective combination of exploration of the global search space and pressure to converge upon the global minima. The quality of a solution is measured using a *fitness* function.

GA-based approaches have been proposed to solve the resource management problem [26]-[28]. However, all of these describe centralized algorithms. Searching for the optimal solution is done on a single node that is selected as the controller. The centralized approach is not suitable for a distributed system. If we assume that all the nodes are supplied with the same battery and have basically the same average processing workload, this would cause relatively faster depletion of energy on the control node, so that it would run out of battery life earlier than other nodes in the network. Such a situation can cause breakdown of the network connectivity in a wireless network. Furthermore, using centralized processing to solve the resource management problem is much less energy efficient than distributed processing. It is known that energy dissipation in a digital system is proportional to the square of supply voltage (Vdd) while the clock frequency is proportional to Vdd [4]. Therefore, by reducing clock frequency and the supply voltage we can reduce the energy dissipation quadratically. Dynamic frequency and voltage scaling (DFVS) is a technique widely supported by processors for mobile computing platforms, such as the Intel XScale embedded processor. We can use DFVS and distributed, or parallel, computing to reduce the overall system energy dissipation. For example, consider a task that takes 1sec and dissipates 1J when run on a single node at full Vdd and clock speed. If the task can be evenly divided between two nodes, by reducing the clock speed and the Vdd of each node to 50% of their original values we can still finish the task in 1sec while the total energy dissipation is only 0.25J.

One of the major characteristics of the Genetic Algorithm is that it is "embarrassingly parallel", in the sense that its workload can easily be evenly distributed among processors, making it an appropriate choice for solving optimization problems in distributed systems. In particular, the Island multi-deme GA is one of the parallel GA models that are widely used [29]. The ODE parameterization and DNA code search software implemented in AFRL/ IFTC under in-house project "Hybrid Architectures for Evolutionary Computing Algorithms" are two good application examples of the Island multi-deme distributed GA model. In this model, the population is divided into several *sub-populations* and distributed on different processors. Each sub-population evolves independently for a few generations, before one or more of the best individuals of the sub-populations *migrate* across processors every few generations. The number of generations between migrations is called an *epoch*.

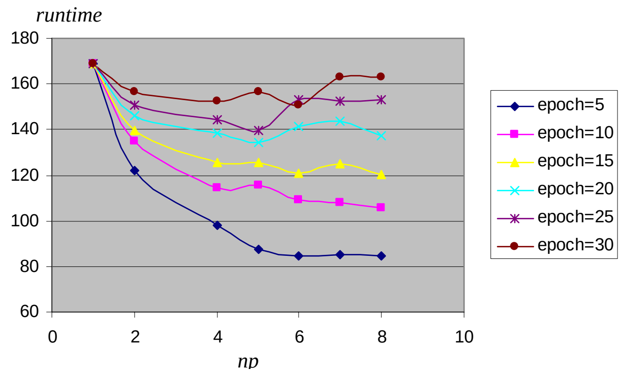
In the ideal situation, by distributing the workload across n processors, the amount of clock time required to converge to the optimal solution becomes $1/n$. In fact, near linear speed-up vs. number of processor nodes has been demonstrated for the in-house application mentioned above. After scaling down the frequency and the voltage of each processor to $1/n$ of their original values, the ratio of the energy dissipation of the distributed processing and the energy of centralized processing becomes $1/n^2$. We call such an ideal situation ideal linear improvement. However, ideal linear improvement is not always available. As the size of each sub-population decreases, the quality of the solution theoretically decreases. Thus there is a minimum population size that should be maintained at each processor to guarantee good performance. Our experiments also show that the degree of migration can be adjusted to help improve the quality of the solutions. There is a close relation between migration and the convergence rate, as well as the quality of solutions.

Some preliminary experiments have been done to assess the impact of the migration speed on the system energy dissipation. In these experiments, we addressed the resource management problem of a small cluster of communication sensor nodes using a distributed multi-deme island model GA similar to the one used in-house at IFTC. In the experiment, we are given J jobs and N nodes. We are interested in finding a way to assign the jobs to different nodes such that the total gain is maximized while the cost is minimized. A pair $<j, n>$ denotes that job j is executed on node n. For each pair $<j, n>$ there is a gain $g_{j,n}$ and a cost $c_{j,n}$. The best resource management scheme is a set S of job and node pairs such that the gain $\sum_{<j,n>\in S} g_{j,n}$ is maximized while the cost $\sum_{<j,n>\in S} c_{j,n}$ is minimized. Note that this is a simple case of resource management. In real application, the gain and cost may not be additive. The user must design a fitness function for the genetic algorithm that depends on the actual cost and gain functions. Using a GA, the individual solution is represented as a string of J values and is denoted as c. The ith entry of this string is represented as c_i which gives the index of the node that is allocated to run the job i. The fitness function is formulated as: $fitness = \lambda_1 \sum_{1\le i\le J} g_{i,c_i} + \lambda_2 \Big/ \sum_{1\le i\le J} c_{i,c_i}$. Fitness based selection probability, single point crossover, and a low level of mutations, and elitism (keeping the best individual from generation to generation) are used in the experiment. The size of the total population is 200. The probability of mating we use is about 80%, i.e. about 20% of the best individuals are kept in the population from generation to generation, and 80% are produced by mating.

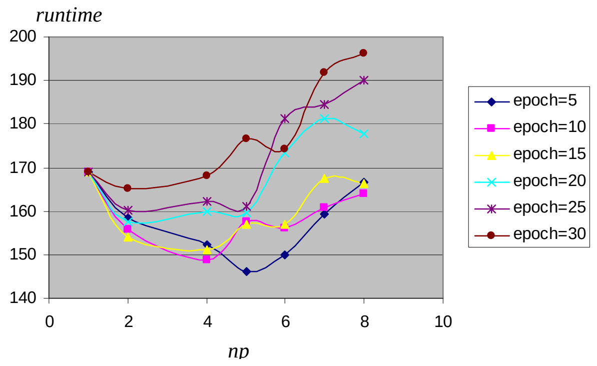
Each sub-population has equal size that depends on the number of processors used. In the experiment, J is set to 100 and N is set to 10. The cost and gain of different job and node pairs are simply set to random variables with a uniform distribution. Note that the search space for this problem is $100!/(90!\times 10!) \approx 8\times 10^{12}$. Therefore, about 4×10^{12} solutions need to be evaluated if exhaustive or simple random search is used. Using GA, we found that only about 3×10^{4} solutions need to be evaluated.

The distributed GA can be configured based on the following parameters: the number of processors (*np*), the size of the sub-population (*sr*), the length of the epoch (*e*), and the number of migrating individuals (*x*). In the experiments, *np* is varied from 1 to 8. Since the population is divided equally, the size of the sub-population is hence varied from 200 to 25. The length of the epoch *e* is varied from 1 to 30 generations. The number of migrating individuals *x* is varied from 1 to 5. In the experiment, we collected the runtime and the energy dissipation of the distributed GA under different configurations. Because the GA is a stochastic program, the runtime and energy is the mean value of 50 runs. There are many migration patterns that could be used, however, we choose to use information broadcasting. During the migration, each node broadcasts its best individual to all of other nodes. Our experiments show that such a migration pattern is more efficient than others. Furthermore, if all of the nodes are within each other's communication range, broadcasting is the most natural and simple way for communication in a wireless network.

To simplify the data analysis we made the following assumptions. These assumptions are based on our observations when running the application on a cluster of computers. The communication is assumed to be blocking. During communication, the CPU is idle. The communication delay is not a function of the number of migrating individuals. We chose to treat communication time as an independent variable, and we varied it from 10% to 90% of the time for a single node to process one generation of 200 individuals. We also assume that DFVS is available on each node so that an improvement in the performance can be converted into the reduction of energy dissipation by running the CPU with slower clock frequency and lower supply voltage. Finally, we assume that the power consumption during migrating is 10% more than the power consumption during computing. This is a conservative assumption, which exaggerates the communication cost. For example, the power consumption of a Lucent ORiNOCO USB Wireless Adapter is 360mA in TX mode and 245mA in RX mode. Assume that the active power of a processor is 300mA, which is the typical power consumption of the XScale processor. The transmission power will be 20% higher than the computing power while the receiving power will be 20% lower than the computing power. Note that during migration, a node is in TX mode for only a short time. It listens to the others for most of the time. Hence, the actual power consumption during the migration period will be much less than the power consumption in TX mode. Furthermore, reducing the clock frequency and supply voltage can only lower the power consumption of the CMOS digital circuit. It will not affect the power consumption of the analog RF circuit. Therefore, in our experiments the communication power is fixed and not affected by DFVS.



(a) 10% communication delay



(b) 90% communication delay

Figure 10 Run time vs. number of processors

We observed that, given enough time, the optimal solution is always found. However, the convergence speeds varied with different configurations. Figure 10 shows the relation between the number of processors and the runtime, for 10% and 90% communication times. Each point on the graph is the average of 50 runs, and each run was terminated when the optimum was found. Using more processors reduces the processing time at first. After a certain point, the reduction becomes very small as Figure 10 (a) shows. If the communication delay is relatively large compared to the processing time, after the number of processors exceeds a certain value, there will be an increase in the runtime as Figure 10 (b) shows. In that case it is obvious that there is an optimal value of *np* that gives the minimum runtime.

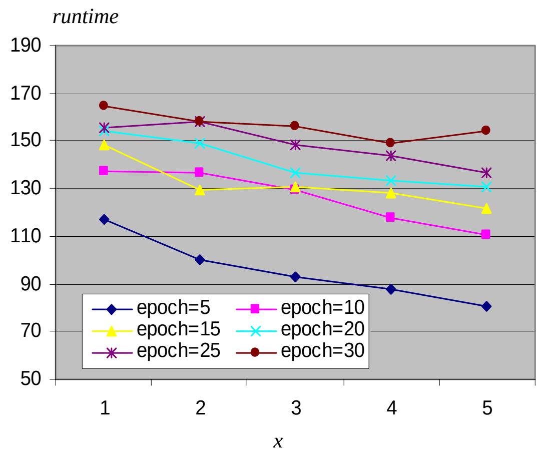
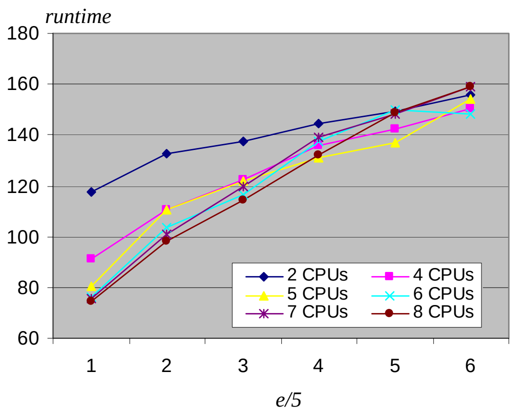


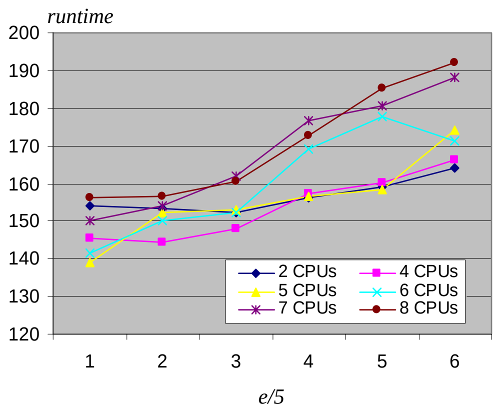
Figure 11 Runtime vs number of migrating individuals

The number of migrating individuals impacts the convergence speed. Our experiments show that the convergence speed can be improved by increasing the number of migrating individuals. There is a nearly linear relation between these two variables as Figure 11 illustrates.

The relation between the length of epoch and the runtime is also nearly linear as Figure 12 shows.



(a) 10% communication delay

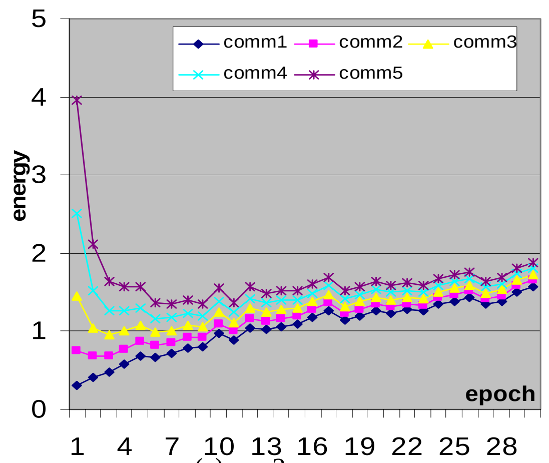


(b) 90% communication delay

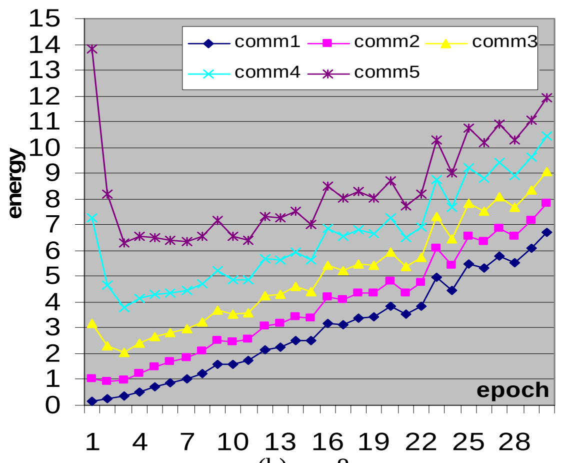
Figure 12 Runtime vs length of epoch

Now we consider the relation between the total energy dissipated on all processors, as a function of epoch length, the communication delay, as well as the number of processors, normalized to the energy dissipation of a centralized GA (i.e. one processor, no communication time).

Figure 13 shows the relation between the length of the epoch and the normalized energy dissipation for systems with different communication delay when *np* = 2 and *np* = 8. In the chart, comm1 ~ comm5 represents the system whose communication delay is 10%, 30%, 50%, 70%, and 90% of the time for a single node to process one generation of 200 individuals respectively. If the communication delay is small, longer epochs always give higher energy. However, if the communication exceeds a certain value, increasing the epoch length will first decrease, and then increase the energy. In that case there is an optimal epoch length that minimizes the energy dissipation.



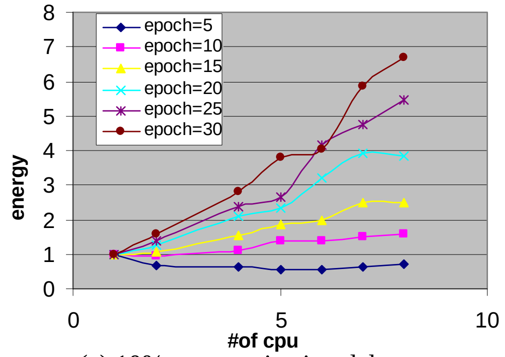
(a) *np*=2



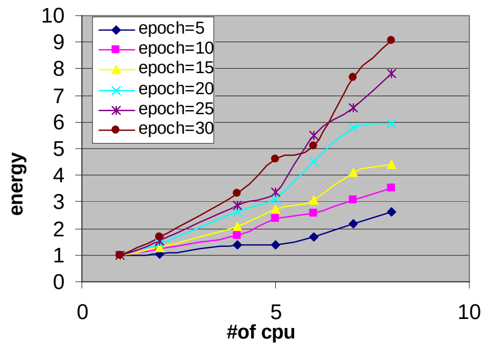
(b) *np*=8

Figure 13 Energy vs length of epoch

Finally, the relation between the energy dissipation and the number of processors *np* is again nearly linear as Figure 14 shows. Note that the energy reported here is the total energy of *np* processors. Each processor consumes 1/*np* of the total energy. By some simple analysis of the data we can find that the total energy of the distributed GA may be higher than that of the centralized GA, but the former results in a longer system lifetime, which is defined as the worst case lifetime of the nodes. For example, for a system with *np* = 8, *e* = 20 and 10% communication delay, the total energy to run the distributed GA is 4 units. Therefore, each processor only dissipates 0.5 units. The worst case lifetime of the node is 2*C*, where *C* is the battery initial capacity. The total energy to run the centralized GA is 1 unit. However, this energy is dissipated on a single node. Therefore, the worst case lifetime of the node is *C*.



(a) 10% communication delay



(b) 90% communication delay

Figure 14 Energy vs number of processors

Our preliminary experimental results show that the configuration of the distributed GA has a significant impact on the convergence speed and the energy dissipation. Further research is needed to explore the relations between population size and migration on convergence speed to find a minimum energy dissipation solution of the resource optimization problem. A wider range of communication times should also be investigated. This would be the first attempt at an in-situ approach to computing a solution to the task assignment problem using an Island Model GA executed on distributed sensor network platform nodes. Success would represent a significant advance since this approach avoids both single control node energy depletion, and costly (non-stealthy) long-haul communication back to a centralized off-network remote processor.

Our future work in sensor network resource management includes two directions.

1) Formulate fuller gain and cost models for a set of representative nodes and tasks. The definition of gains and costs are very application specific. In this work, we are going to use the cognitive weather sensor network that is developed under an AFRL/IFTC project as target system. The set of tasks running on each node in the system will be characterized. For example, each task brings in, processes, and communicates some information about the environment. The criticality of successfully gathering, processing and communicating this information can be used as the gain of the task. Empirical definitions or theoretical definitions (for example, entropy based definitions) of the gain will be tested. Techniques for adaptively updating the gain and the cost of the tasks will also be investigated.

2) Design an energy efficient implementation of an island model GA solver to handle the task assignment optimization problem using nodes in the distributed mobile platform. We are going to study the impact of the population size and migration rate to the computing and communication energy dissipation and find the best trade-off. Because of the stochastic nature of the GA, models such as Markov chain, a Markov decision process will be used. This task is a fundamental study of the performance and energy optimization of the distributed GA. The results of this study should be useful in a variety of military and non-military applications.

5 CONCLUSIONS

We studied low power techniques for distributed systems, especially sensor networks. At the node level, we focused on energy aware programming. Our results show that, due to the special architecture of embedded processors such as the XScale processor, some simple program transformations (e.g. using integer vs floating point calculations) can significantly improve the software performance and energy dissipation. Our study also shows that the traditional scheduling algorithms do not work well with power management in a sensor node with periodic incoming tasks. Two heuristic algorithms are proposed which dissipate less energy. At system level, we focused on energy aware resource management. We propose to formulate the resource management problem as an optimization problem and solve it using a distributed Genetic Algorithm. We believe that the Genetic Algorithm will be a good candidate to solve the optimization problem in a distributed sensor network, because its workload can be evenly distributed, which enables energy balancing and performance improvement. We also experimentally analyzed the performance and the energy of the distributed GA, and found that they are closely related to several GA configuration parameters.

6 REFERENCES

[1] ISTAG, “*Ambient Intelligence: From Vision to Reality*,” Sept. 2003, ftp://ftp.cordis.lu/pub/ist/docs/istag-ist2003_draft_consolidated_report.pdf.

[2] I. F. Kyildiz, S. Weilian, Y. Sankarasubramaniam and E. Cayirci, “A Survey on Sensor Networks,” *IEEE Communications Magazine*, Volume 40, Issue 8, pp. 102-114, Aug. 2002.

[3] E. R. Post and M. Orth, “Smart Fabric, or Wearable Computing,” *Proc. First Int’l Symp. Wearable Computers*, pp. 167-168, Oct. 1997.

[4] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. on Design Automation of Electronic Systems*, Vol. 1, No. 1, pp. 3-56, 1996.

[5] L. Benini, G. de Micheli, “System-Level Power Optimization: Techniques and Tools,” *Proc. International Symposium on Low Power Electronics and Design*, pp. 288-293, Aug. 1999.

[6] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, “Energy-Efficient Communication Protocols for Wireless Microsensor Networks,” *Proc. Hawaiian Int’l Conf. On Systems Science*, Jan. 2000.

[7] J-H Chang and L. Tassiulas, “Energy Conserving Routing in Wireless Ad Hoc Networks,” *Proc. of Infocom*, pp. 22-31, Apr. 2001.

[8] A. Iranli, H. Fatemi, and M. Pedram, "A Game Theoretic Approach to Dynamic Energy Minimization in Wireless Transceivers," *Proc. of Int'l Conference on Computer Aided Design*, Nov. 2003, pp. 504-509.

[9] H. Su, P. Qiu and Q. Qiu, “ESACW: An Adaptive Algorithm For Transmission Power Reduction In Wireless Networks,” *to appear on Proc. of ISLPED*, Aug. 2004.

[10] Q. Qiu, Q. Wu, M. Pedram, "Stochastic Modeling of a Power-Managed System: Construction and Optimization," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 194-199, Aug. 1999.

[11] Q. Qiu, Q. Wu, M. Pedram, “Dynamic Power Management of Complex Systems Using Generalized Stochastic Petri Nets,” *Proceedings of the Design Automation Conference*, pp. 352-356, Jun. 2000.

[12] Q. Wu, M. Pedram, Q. Qiu, “Dynamic Power Management in a Mobile Multimedia System with Guaranteed Quality-of-Service,” *Proceedings of the Design Automation Conference*, Jun. 2001.

[13] V. Tiwari, S. Malik and A. Wolfe, “Power analysis of embedded software: a first step towards software power minimization,” , *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 2, Issue 4, pp. 437-445, Dec. 1994.

[14] A. Sinha, N. Ickes and A.P. Chandrakasan, “Instruction level and operating system profiling for energy exposed software,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 11, Issue 6, pp. 1044-1057, Dec. 2003.

[15] http://platformx.sourceforge.net/home.html

[16] http://pasta.east.isi.edu/

[17] ftp://download.intel.com/design/intelxscale/XScaleDatasheet4.pdf

[18] http://www.eecs.umich.edu/~panalyzer/

[19] http://www.cse.psu.edu/~mdl/software.htm

[20] http://carlsberg.mit.edu/JouleTrack/

[21] M. Srivastava, A. Chandrakasan. R. Brodersen, “Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Transactions on VLSI Systems*, Vol. 4, No. 1, pp. 42-55, 1996.

[22] C.-H. Hwang and A. Wu, “A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation,” *Proc. of the Intl. Computer-Aided Design*, pp. 28-32, 1997

[23] G. A. Paleologo, L. Benini, et.al, “Policy Optimization for Dynamic Power Management”, *Proceedings of the Design Automation Conference*, pp.182-187, Jun. 1998.

[24] Qinru Qiu, Q. Wu, M. Pedram, “Stochastic Modeling of a Power-Managed System: Construction and Optimization,” *IEEE transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 20, No. 10, pp. 1200-1217, Oct. 2001.

[25] G. De Micheli, “Synthesis and Optimization of Digital Circuits,” *Mcgraw-Hill*, 1994.

[26] M. Munetomo, Y. Takai, Y. Sato, “A Stochastic Genetic Algorithm for Dynamic Load Balancing in Distributed Systems,” *IEEE International Conference on Systems, Man and Cybernetics*, pp. 3795-3799, 1995.

[27] A. Talukder, R. Bhatt, L. Chandramouli, S. M. Ali, R. Pidva, S. Monacos, “Autonomous Resource Management and Control Algorithms for Distributed Wireless Sensor Networks,” *ACS/IEEE International Conference on Computer Systems and Applications*, pp. 19, 2005.

[28] W.J. Song, B.H. Ahn, “Distributed Power Control Using the Simultaneous Mutation of Genetic Algorithms in Cellular Radio Systems,” *International Conference on Information Technology: Coding and Computing*, pp. 361-364, 2002.

[29] E. Cantu-Paz, “A Survey of Parallel Genetic Algorithms,” *Calculateurs Paralleles, Reseaux et Systems Repartis*, Vol. 10, No. 2.